Evolution of the Distributed Computing Model of the CMS experiment at the LHC

2012 J. Phys.: Conf. Ser. 396 032053

(http://iopscience.iop.org/1742-6596/396/3/032053)

Journal of Physics: Conference Series **396** (2012) 032053 doi:10.1088/1742-6596/396/3/032053

Evolution of the Distributed Computing Model of the CMS experiment at the LHC

C.Grandi[1]; B. Bockelman[2]; D. Bonacorsi[1]; G. Donvito[3]; D.Dykstra[4]; I. Fisk[4]; J.Hernandez[6]; S.Metson[6]; I.Sfiligoi[7]; S. Wakefield[8]

[1]INFN-Bologna & University of Bologna; [2]University of Nebraska, [3]INFN-Bari; [4]FNAL; [5]CIEMAT; [6]University of Bristol; [7]UCSD; [8]Imperial College of London

Claudio.Grandi@cern.ch

Abstract. The Computing Model of the CMS experiment was prepared in 2005 and described in detail in the CMS Computing Technical Design Report. With the experience of the first years of LHC data taking and with the evolution of the available technologies, the CMS Collaboration identified areas where improvements were desirable. In this work we describe the most important modifications that have been, or are being implemented in the Distributed Computing Model of CMS. The Worldwide LHC computing Grid (WLCG) project acknowledged that the whole distributed computing infrastructure is impacted by this kind of changes that are happening in most LHC experiments and decided to create several Technical Evolution Groups (TEG) aiming at assessing the situation and at developing a strategy for the future. In this work we describe the CMS view on the TEG activities as well.

1. Introduction

In the following sections we describe the modifications that have been made and that are planned to be made to the CMS distributed computing system, with respect to what is described in the Computing Model [1] and in the Computing Technical Design Report [2] prepared in 2005. The main areas that will be described are the Workload Management (Section 2) and the Storage and Data Management (Section 3). We will refer, when relevant, to the activities of the Technical Evolution Groups created by the WLCG Project to discuss the evolution of the computing infrastructure for the LHC experiments. Security and Database issues are discussed when impacting the distributed computing model, in the aforementioned sections.

2. Workload Management

2.1. Overall architecture

The baseline workload management system presented in the CMS Computing TDR [2] is based on a standard push-model. The main assumption was that all processing jobs are sent to the site hosting the data. The system was based on a thin CMS layer sitting on the User Interface (UI) and exploiting the tools provided by the Grid projects active at that time (gLite-WMS and the BDII, the gLite Information System). The advantage of this model was the intrinsic scalability that could be achieved

by replication of the Workload Management System (WMS) servers. The schema is shown in Figure 1.A: jobs are sent by the WMS to the Computing Elements (CE) at the sites.

In the same document CMS presented a possible evolution based on a 2-layer pull model shown in Figure 1.B. This model foresaw a Global Task Queue and Local Task Queues at CMS sites. The Global Task Queue allowed the implementation of policies that could be used to prioritize the work CMS-wide. The Local Task Queue was actually controlling the evolution of the job execution at the site. An important concept was that the harvesting of resources was separated from the actual job execution: a service running at the site had the responsibility to request the job slots on the local batch system and execute a process able to contact the Local Task Queue to get the actual job to be executed. The main advantages of this model would have been the independence from an information system and the possibility to implement CMS-wide policies. On the other hand it required the Global Task Queue to scale to the size of the whole CMS and the need for the sites to maintain CMS specific services.

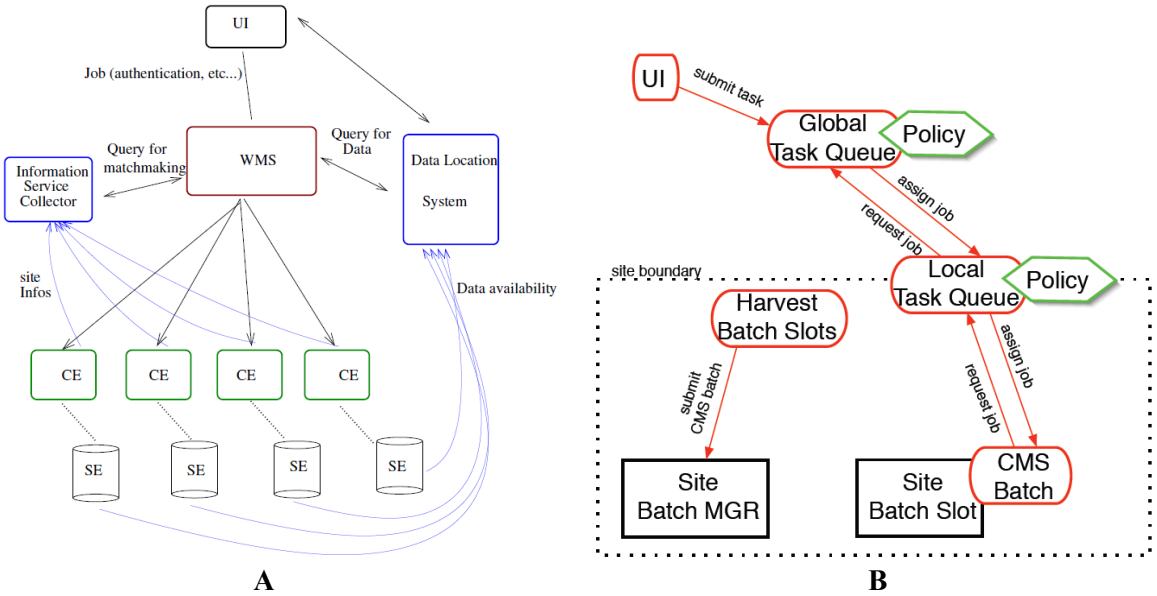

Figure 1: Workload management schemas as foreseen by the CMS Computing TDR [2]: A: The baseline mechanism used by CMS in the first years of data taking. B: The foreseen evolution to a pilot based model.

The architecture implemented in the last years by CMS is an evolution of the second one. The main difference is that the batch slot harvesting and the Local Task Queues are moved out of the site boundaries, so there is no extra load for the CMS site administrators.

The system uses the glidein-WMS service [3] which heavily relies on Condor [4]. A schematic view is shown in Figure 2. The main components of the glidein system are:

- The *glidein frontend* that receives the user job submission requests keeps the queue of the jobs, implements the global policies and dispatches the jobs to the execution hosts.
- The *glidein factory* that has the responsibility to harvest batch slots. It receives requests from the frontends and submits glidein jobs to the CEs.
- The *glidein* itself that is the pilot job that goes into execution on the Worker Node (WN) at the site. It starts a Condor *startd* process on the WN that contacts a condor *collector* process on the frontend to get a user job to be executed.

The CMS specific components of the system are:

- The *Global Task Queue* that accepts task execution requests from the *Request Manager* (not shown in the figure) that in turns receives the request either from a web GUI or a CLI.

- The *WMAgent* that translates the task execution request into a set of jobs properly prepared to be submitted to the distributed infrastructure. The WMAgent has a *Local Task Queue* where it is possible to throttle the dispatching of jobs to the subsequent steps of the submission chain.
- The *CMS job* itself that is the user executable that uses the locally installed software to process the events.

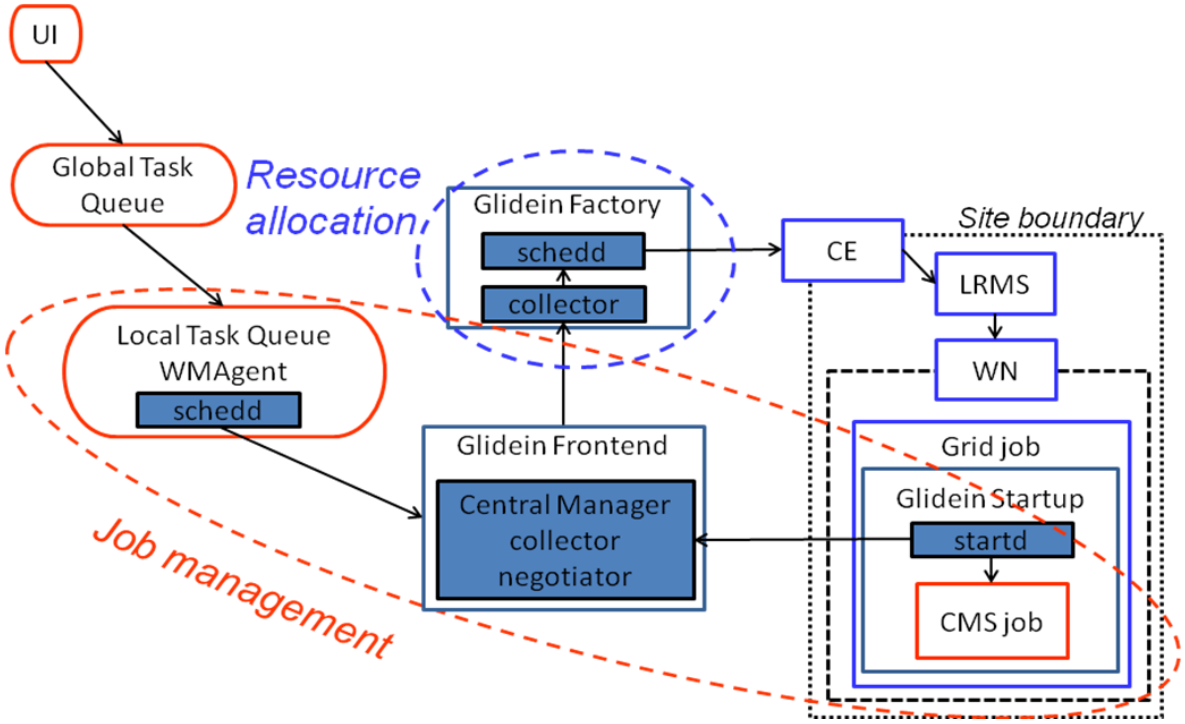

Figure 2: Schematic view of the glidein-based CMS workload management system.

As it can be seen in Figure 2 the job management and the resource allocation chains are separated, allowing greater flexibility.

2.2. Security issues

The current Grid model relies on the fact that authentication, authorization and auditing (AAA) is done on the site gateway i.e. on the CE. In a pull model this is still true for the resource allocation phase but not for the job management; in this case the AAA must happen on the WN after the user job is fetched by the Condor *startd*. A few years ago the *gLExec* [5] tool was developed. *gLExec* allows the extra AAA step letting the site managers have control not only on the resource allocation, but also on the job execution phase. *gLExec* also provides another important feature. The execution of the user job (payload) happens with a different UNIX identifier, thus isolating the execution environment of the pilot and the payload. This guarantees that a malicious payload cannot operate as if it was the pilot (e.g. downloading other user jobs) and this is deemed extremely important for CMS.
CMS has been successfully using *gLExec* on sites supporting it and recommends that it be deployed on the whole infrastructure. This is now likely to happen soon since the WLCG Security and Workload Management TEGs recommended the use of *gLExec* for pilot job execution.

2.3. Medium and long term improvements

In section 2.1 we described how the resource allocation is separated from the job management. Still the latter happens on a job by job basis: the *glidein factory* has the responsibility to keep the number of pilot jobs at the different sites at a level sufficient to execute the CMS jobs continuously interacting

with the CEs to get the needed job slots. Since this phase actually consists in submitting identical jobs, the WLCG WM TEG recommended to expand the CE interface to support the request to "constantly keep N identical jobs queued at a given queue until a given condition is satisfied (e.g. no work to be done)". This is indeed a useful feature and CMS recommends that this is implemented soon.

CMS, as other HEP experiments, demonstrated that there is an improvement in memory usage when processing events in parallel on many cores within the same job instead of running independent jobs each using a single core. This is measured to be about 25% when running a typical CMS job on 8 cores compared with 8 single core jobs [6]. The reason is that a significant fraction of the job memory is occupied by the detector geometry, run conditions, alignment and calibrations, etc... that are common to all events and can be shared among the processes analyzing different events on different cores using the Copy on Write (CoW) feature provided by the Linux kernel. Other improvements concern the overall reduction in the number of jobs handled by the workload management system and the reduction of the complexity of the so-called "asynchronous merge" step (when the results of independent jobs are combined).

For this reason CMS is pushing for a modification of the resource allocation model that allows getting multi-core job slots instead of single core slots. It should be noted that in principle CMS can run on multi-core resources by letting the *glidein* to fetch multiple single-core jobs. This doesn't provide any direct improvement in resource usage, but can be used to run jobs that cannot be parallelized at the time the resource allocation model should change.

The current implementation in *WMAgent* follows, for simplicity reasons, a "whole-node" approach: the WMAgent wrapper looks in `/proc/cpuinfo` to get the number of available cores and starts a corresponding number of processes. This is an implementation detail and there is no reason why the number of processes to start cannot be predefined and passed as argument to the *WMAgent* wrapper, making it a "multi-core" rather than "whole-node" job. Nevertheless CMS concentrated its efforts on the "whole-node" operation mode because it is more efficient than scheduling arbitrary "multi-core" jobs, due to reduced complexity at the batch scheduling level and because it eliminates interference among users and the need to impose limits e.g. on memory usage. Furthermore it is a natural direction for Cloud and virtualized machine scheduling.

Recently the WLCG WM TEG recommended adopting a generic "multi-core" approach, where the total number of cores on a machine can be allocated to several "multi-core" jobs. This is also acceptable for CMS even though mechanisms to inform the job about the amount of resources allocated need to be developed and deployed at all sites.

There is increasing interest in the HEP community about the utilization of a Cloud interface to access distributed resources. CMS is happy with the current interface as the allocation of a "whole-node" job slot would offer all the needed functionalities. Switching to a Cloud-like interface may be considered, if the decision is that the whole infrastructure goes in that direction. The option is acceptable by CMS in case resources are provided in a way that does not jeopardize the efficiency (e.g. resources should not be in a *De-Militarized Zone* (DMZ) and should continue to have efficient access to the local storage). CMS already tried to use commercial clouds [7] and decided they may be taken into account to absorb simulations peaks but not for analysis for which the cost would be too big. This is currently not considered as a mainstream activity, though.

A particular aspect that is often treated together with Clouds is virtualization. Actually virtualization exists independently of Clouds and may offer useful functionalities independently from the use of a Cloud interface. Besides the possibility to use virtualization to run multiple relatively lightweight services on new, powerful multi-core machines, it provides flexibility to sites in the customization of resources for the supported Virtual Organizations (VO). The position of CMS concerning virtualization is that it is acceptable to have resources allocated as virtual machines but the decision is left to sites: sites are free to provide virtualized resources if they can be used in a transparent way without significant performance degradation.

3. Storage and Data Management

3.1. Data distribution

The original data distribution model foreseen in the Computing TDR is shown in Figure 3 and it is a hierarchical model inherited from MONARC [8]. After the initial operations at the Tier0 at CERN, the experiment data is moved to the Tier1s for archival and for organized processing, and to the Tier-2s connected to each Tier1 for user analysis. Monte Carlo simulated data is produced at Tier2s and Tier1s, and stored at Tier1s; then they follow the same path of the experiment data. Copies from Tier2s of different regions had to go through the respective Tier1s.

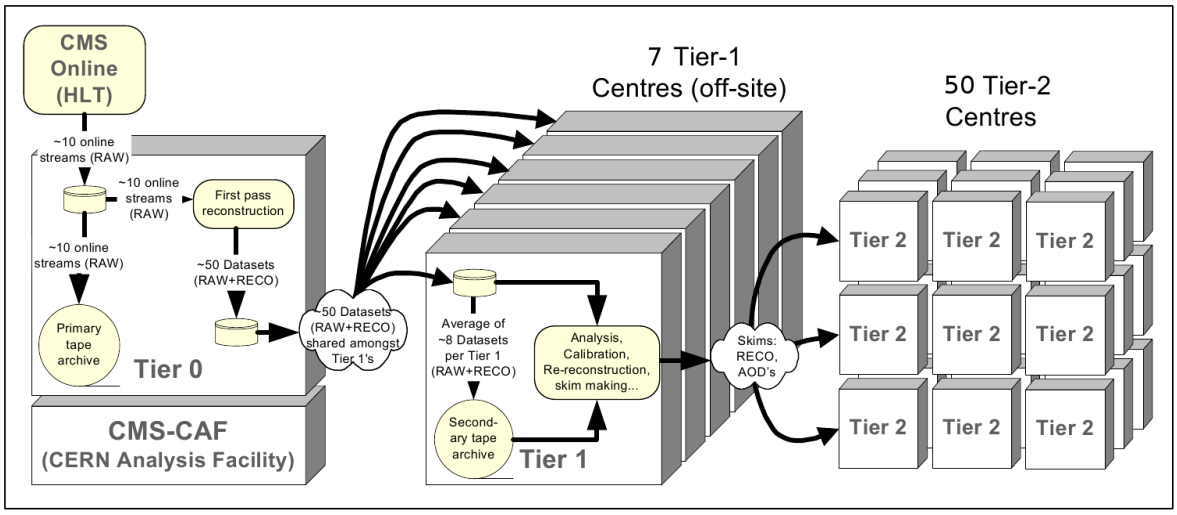

Figure 3: The CMS Data Distribution Model as foreseen in the Computing TDR

The first important modification to the original model has been the implementation of the full mesh connecting every Tier2 to every Tier1 and in a second phase to other Tier2s as well. This happened in 2009-2010. In Figure 4 it is possible to see how in 2010 the Tier2-Tier2 traffic represented about 20% of the total volume.

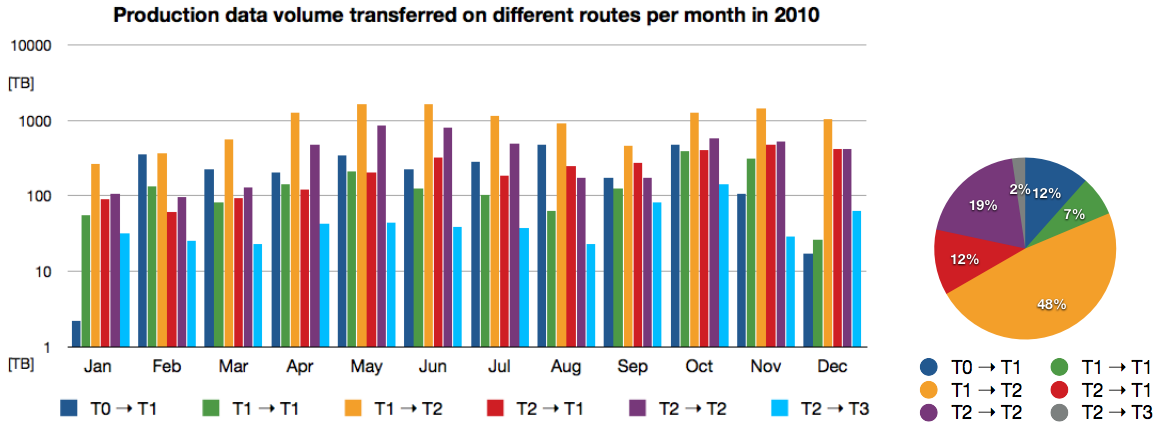

Figure 4: CMS traffic on the different route categories in 2010

It can also be noted how the implementation of the full mesh moved some of the Tier0-Tier1 traffic to Tier1-Tier1 as PhEDEx [9], the CMS transfer system, could reroute transfers to Tier1s using other Tier1s as a source.

One of the most critical issues experienced by CMS in the first years of data taking has been the management of data replication at Tier2s for analysis support. As already mentioned in [10] many of the replicated datasets were seldom accessed, causing a sub-optimal utilization of the disk resources at the sites. A data popularity service [11] based on the information collected by the CMS Dashboard [12] has been developed. This service is helping the operators in the replica management.

3.2. Storage federations

The decreasing cost and increasing reliability of network resources had an important impact on the evolution of the CMS data management architecture as it is now possible to contemplate in the model also access to remote data (i.e. not residing on the same LAN of the analysis process).

While it is acknowledged that data placement is still the driving procedure for efficient data processing, CMS individuated a few use cases where remote access could play a significant role:

1. In case of unavailability of the local copy of a file the user job can transparently access a remote replica, thus increasing the processing efficiency.
2. With low I/O processes, i.e. visualization programs.
3. To address site congestion, when the available copies of a dataset are at overloaded sites.
4. To increase the utilization of CPU power at sites where proper data management is not possible, e.g. at Tier-3 sites.

CMS demonstrated how using *xrootd* [13] as a fall back protocol at sites it was possible to transparently access remote files when they were not available locally through the predefined protocol (that could be xrootd itself). Through a hierarchical structure of *redirectors* it is possible in *xrootd* to

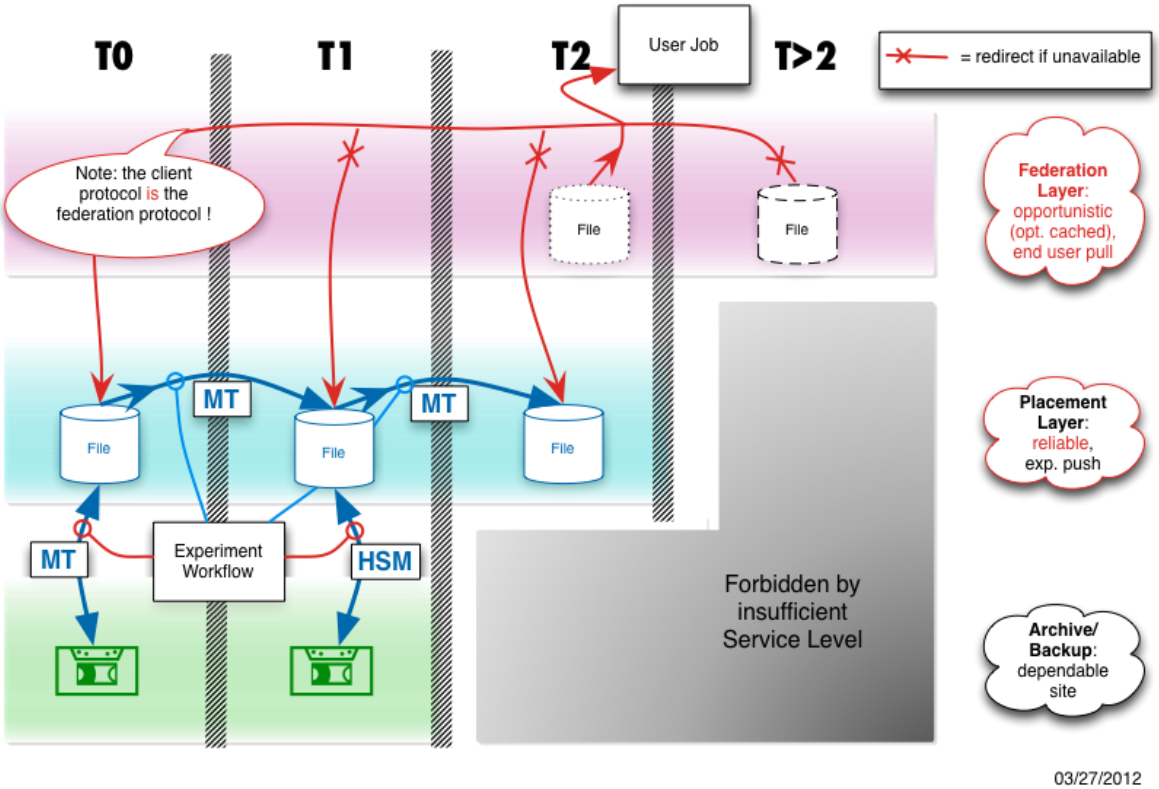

Figure 5: The Data Management model proposed by the WLCG Storage and Data Management TEGs. It includes an archive/backup layer that provides reliable permanent data storage, a data placement layer responsible for providing client access to datasets and a Federation layer that increases data availability.

federate many site storage services. This methodology has recently been proposed by the Storage and Data Management WLCG TEGs as an additional layer for data management (see Figure 5).

CMS is already using *xrootd* at many sites to provide fall back in case of error in accessing local files and it has been integrated in the visualization programs (points 1. and 2. above). There is already a strategy for addressing site congestion (point 3.) using the *glidein-WMS*, and this will be evaluated in the next months. Performance and scale tests are ongoing to support Tier3 sites without local CMS-managed storage (point 4.).

The main issue with remote access is the scalability not only of the network but also of the storage resources since every access implies full AAA that is in general much lighter in case of local access. For this reason proper monitoring and resource throttling tools must be available at the same time this model is widely adopted. Furthermore care has to be taken when serving data on a Hierarchical Storage Management (HSM) system as direct access to tape servers must be controlled. This is actually a more general issue related to HSM protection that is addressed in the next section.

The choice of *xrootd* has been driven by its availability and readiness for CMS needs but in future it is not excluded to replace it with more widely used protocols like *http* in case they offer the same functionalities.

3.3. Partition Tier1 tape and disk resources

CMS is currently using a unique T1D0 storage class at Tier1s with a huge buffer and access to the tape is done transparently. CMS has demonstrated that this model is working well and thanks to the transparency in the access puts relatively small load on the CMS operators. On the other hand it creates additional constraints to the overall computing model. The main reason is that the tape system must be protected from uncontrolled recalls; for this reason user analysis and remote access to data at Tier1s is currently limited. Furthermore there is not much flexibility in deciding what goes to tape and what does not, and when to start migrating files to tape (it is known that the tape system are more efficient if operations are done in bulk).

Following a TEG recommendation, CMS is speeding up the process of splitting tape and disk storage into a T1D0 system with a relatively small buffer and a bigger T0D1 system that will serve data to jobs running on the computing farm. Data will have to be explicitly replicated to the T0D1 system to process them and mechanisms to control the occupancy of the T0D1 partition will need to be created. This would require modifications in the data distribution and data access systems.

The completion of this process would provide the needed protection for the mass storage system and would allow opening Tier1s to analysis and remote access.

3.4. Non-event data distribution

CMS needs to move non-event data. In the Computing TDR it was already foreseen to have caches at the sites for the distribution of conditions data and to provide efficient access to them from running jobs. A network of *squid* servers was deployed for that purpose (see Figure 6) and the *http* caching mechanisms optimized for the scope [14]. Recently the *squid* network has been used also for the distribution of small files needed by Monte Carlo generators that were more efficiently accessed from the *squid* cache than in the CMS software area. More recently, following the recommendation of the WLCG TEGs, CMS started to use CernVM File System (CVMFS) [15] that is also based on *squid* caches to distribute CMS software to sites. At a later stage the same procedure will be used to transfer other relatively small files like those needed by the pilot framework. The network of *squid* caches will represent an even more significant component of the CMS data management infrastructure.

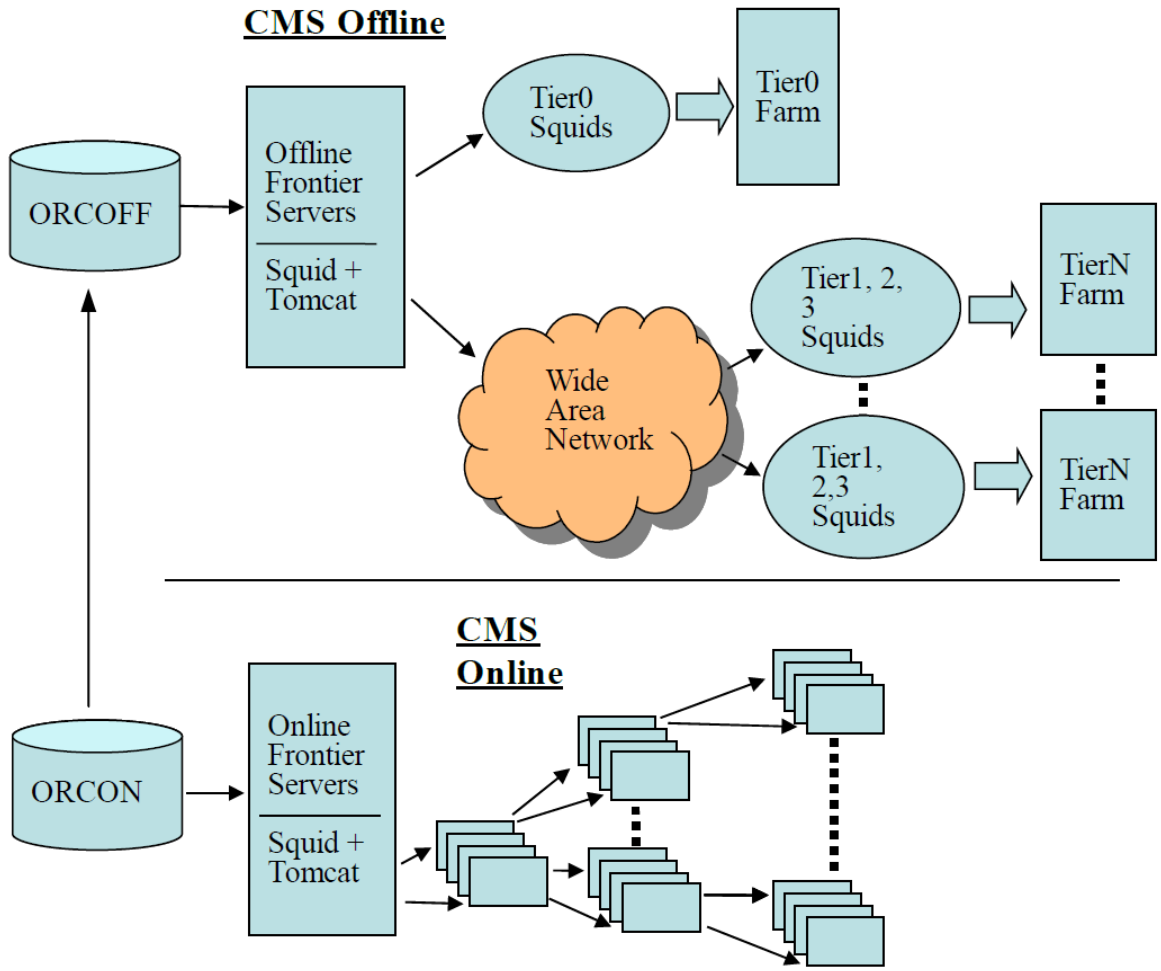

Figure 6: The caching mechanism used for CMS database distribution [14]

4. Conclusions

In this paper we described recent activities concerning the evolution of CMS Distributed Computing Model that potentially will make the infrastructure more scalable and operable.

The evolution of the 2-layer pull model initially foreseen in the Computing TDR for workload management provides CMS with the needed flexibility to efficiently use computing resources and address modifications in the resource allocation model. The possible migration to a Cloud resource allocation model could be easily accommodated.

The modification of the CMS software framework to exploit multi-core machines guarantees a more efficient use of resources. CMS recommends the adoption of a whole-node resource allocation model.

CMS abandoned the hierarchical data distribution model initially adopted a few years ago. Data transfers efficiently happen among all Tier1 and Tier2 sites. The implementation of a *Federation Layer* on top of the existing *Data Placement* and *Archival Layers* makes the data access even more efficient, increasing the data availability for CMS users.

The distribution of all non-event data, including CMS software and other files that need to be available at all sites, exploits a network of caches that has proven to work efficiently for the conditions and calibration databases.

References

[1] C.Grandi, D.Stickland, L.Taylor et al. "The CMS Computing Model" CERN-LHCC-2004-035/G-083 (2004)
[2] The CMS Collaboration, "CMS Computing Technical Design Report", CERN-LHCC-2005-023 (2005)
[3] I.Sfiligoi et al. "The Pilot Way to Grid Resources Using glideinWMS", WRI World Congress on Computer Science and Information Engineering, ISBN: 978-0-7695-3507-4 (2009)
[4] D.Thain, T.Tannenbaum and M.Livny "Distributed computing in practice: the Condor experience" Concurrency and Computation: Practice and Experience, vol.17, no.2-4, pp.323-356 doi:10.1002/cpe.938 (2005)
[5] D.Groep, O.Koeroo, G.Venekamp "gLExec: gluing grid computing to the Unix world" *J. Phys.: Conference Series* **119** 062032. doi:10.1088/1742-6596/119/6/062032 (2008)
[6] J.Hernandez et al. "Multi-core processing and scheduling performance in CMS", presented at the Conference on Computing in High Energy and Nuclear Physics (CHEP12), New York, USA (2012)
[7] D.Evans et al. "Using Amazon's Elastic Compute Cloud to dynamically scale CMS computational resources" *J. Phys.: Conf. Ser.* **331** 062031 doi:10.1088/1742-6596/331/6/062031 (2011)
[8] M.Aderholz et al., "Models of Networked Analysis at Regional Centres for LHC xperiments (MONARC) - Phase 2 Report", CERN/LCB 2000-001 (2000)
[9] T.Wildish et al. "From toolkit to framework - the past and future evolution of PhEDEx", presented at the Conference on Computing in High Energy and Nuclear Physics (CHEP12), New York, USA (2012)
[10] C Grandi et al "CMS Distributed Computing Integration in the LHC sustained operations era" *J. Phys.: Conf. Ser.* **331** 062032 doi:10.1088/1742-6596/331/6/062032 (2011)
[11] D.Giordano et al "Implementing data placement strategies for the CMS experiment based on a popularity model", presented at the Conference on Computing in High Energy and Nuclear Physics (CHEP12), New York, USA (2012)
[12] J.Andreeva et al. "Dashboard for the LHC experiments", Conference on Computing in High Energy and Nuclear Physics (CHEP07), Victoria, Canada (2007)
[13] Xrootd: http://xrootd.slac.stanford.edu/
[14] D.Dykstra and L.Lueking "Greatly improved cache update times for conditions data with Frontier/Squid" *J. Phys.: Conf. Ser.* **219** 072034 doi:10.1088/1742-6596/219/7/072034 (2010)
[15] J.Blomer, T.Fuhrmann "A Fully Decentralized File System Cache for the CernVM-FS", 19th IEEE International Conference on Computer Communications and Networks (ICCCN); doi: 10.1109/ICCCN.2010.5560054 (2010)